SMI6U05 : Bases de données
Le langage SQL

Meryem Billami et Luigi Santocanale
LIS, Aix-Marseille Université

Contenu basé sur le cours de Rémi Eyraud

Plan

Plan

Jointure et clés étrangères

```
Chanson(id_chanson, titre, auteur, ...)
Artiste(id_artiste, nom, prenom, ...)
```

Chanson

id_chanson	titre	auteur
1	Ne me quitte pas	2
2	Que je t'aime	1
3	Les Lacs du Connemara	3
4	Je serais là	1

$\bowtie_{\texttt{auteur = id\_artiste}}$

Artiste

id_artiste	nom	prenom
1	Hallyday	Johnny
2	Brel	Jacques
3	Sardou	Michel

$=$

Chanson

id_chanson	titre	auteur	id_artiste	nom	prenom
1	Ne me quitte pas	2	2	Brel	Jacques
2	Que je t'aime	1	1	Hallyday	Johnny
3	Les Lacs du Connemara	3	3	Sardou	Michel
4	Je serais là	1	1	Hallyday	Johnny

Plan

*Historique de **SQL***

Dans ce cours : **SQL**:2003 (sans aspects objets).

Plusieurs normalisations :

- **SQL** 1 (1986) : première norme
- **SQL** 2 (1992) :
  SQL 1 + nouvelles instructions
  (par exemple : `JOIN`)
- **SQL** 3 (1999) :
  SQL 2 + approche orienté objet
- **SQL**:2003 :
  SQL 3 + quelques modifications mineures
  (ex : **SQL**/XML)
- **SQL**:2008 et **SQL**:2011 : quelques modifications

Les classes d'instructions

- Les instructions **SQL** sont découpées en plusieurs classes.
- Chaque instruction appartient à une classe fonctionnelle.
- Classe fonctionnelle :
  ensemble de fonctions logiquement reliées.

- **SQL** 2 :
  - Langage de manipulation de données (LMD)
  - Langage définition de données (LDD)
  - Langage de contrôle de données (LCD)

*Les classes d'instructions : **SQL** 3*

1. Instructions de connexion.
   Ouvrir et fermer une connection à une base.
   `CONNECT`, `DISCONNECT`, ...
2. Instruction de contrôle.
   Contrôler l'exécution d'un groupe d'instructions
   `CALL`, `RETURN`, ...
3. Instructions sur les données (ex-LMD).
   Effet persistant sur les données.
   `SELECT`, `INSERT`, `UPDATE`, `DELETE`, ...

*Les classes d'instruction : **SQL** 3*

4. Instruction de diagnostic.
   Accès à des informations de diagnostic d'erreurs ou d'exceptions.
   Aussi : initient les levées d'erreurs.
   `GET DIAGNOSTICS`.

5. Instruction sur les schémas (ex-LDD).
   Effet persistant sur les schémas.
   `ALTER`, `CREATE`, `DROP`, ...

6. Instruction de session.
   Contrôlent les paramètres d'une session de connexion.
   `SET`, `SET CONSTRAINTS`,...

7. Instruction de transaction.
   Fixent le début et la fin d'une transaction.
   `COMMIT`, `ROLLBACK`,...

*Dialectes procéduraux **SQL***

- **SQL** n'est pas procédural :
  - pas d'instruction conditionnelle
  - pas d'itération, pas de boucles

  C'est un langage déclaratif relationnel.
- Gestion et traitement des erreurs :
  procédures sont bien pratiques.
- Au sein des SGBDR : des langages définis pour ces structures de contrôle (pas normalisés), proche des constructions **SQL**.
- Exemples :
  - PL/SQL (Procedural Language/SQL) d'Oracle,
  - triggers en **SQLite3**
  - interfaçage par un autre langage de programmation

Plan

*Les instructions **SQL** sur les schémas*

- Langage de Définition des Données (LDD)
- Créer, modifier ou supprimer des tables, des contraintes.
- Norme vs **SQLite3**
- Le lien qui vous sauvera la vie en TP
  `https://www.sqlite.org/docs.html`

Création d'une table, premier exemple

```
Film(numfilm, titre, annee, mois, realisateur)
```

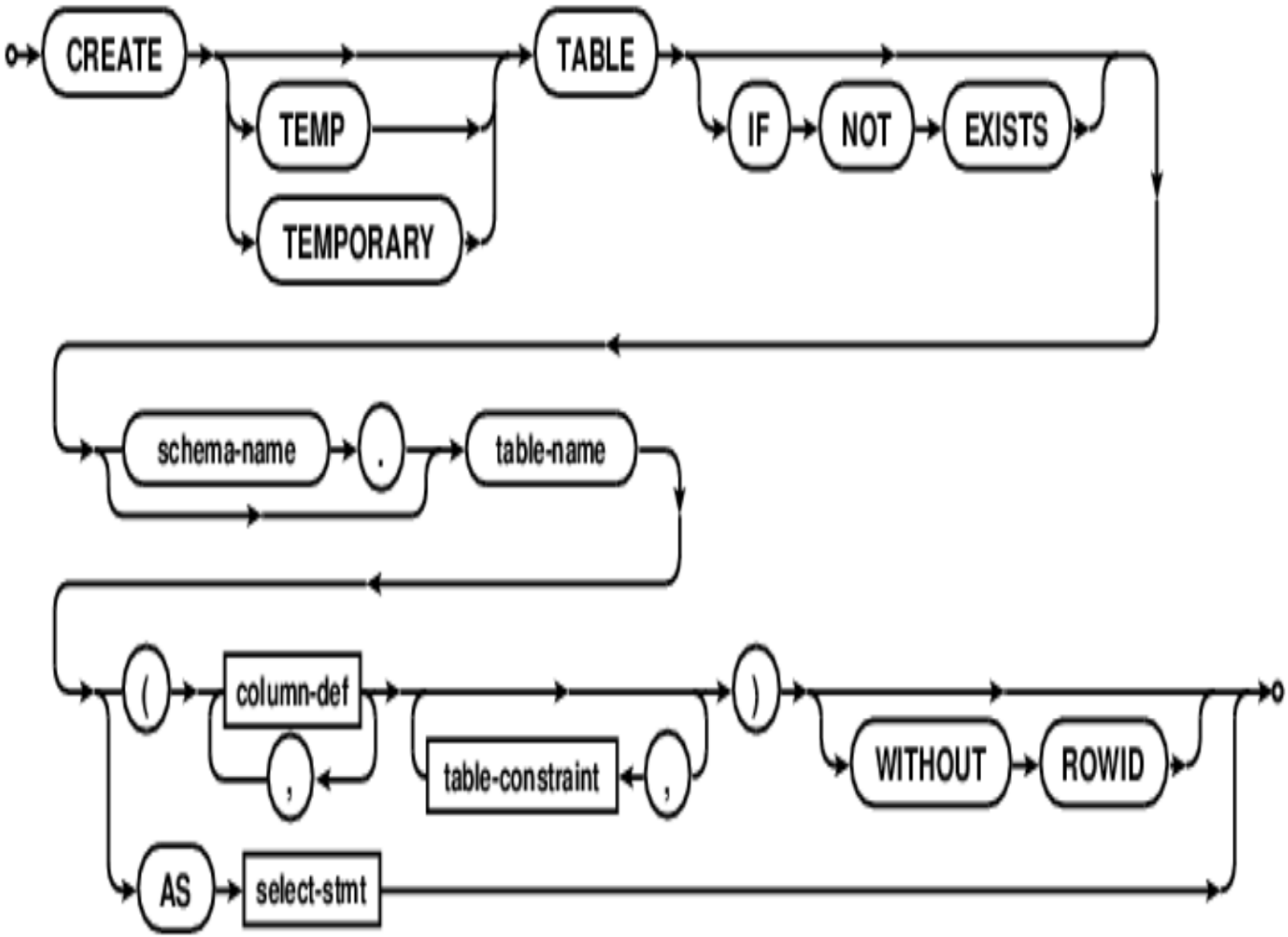
```
CREATE TABLE Film (
       numfilm INTEGER PRIMARY KEY,
       titre TEXT,
       annee INTEGER,
       mois INTEGER,
       realisateur TEXT
      )
```

Création d'une table, deuxième exemple

```
Film(numfilm, titre, annee, mois, realisateur)
```

```
CREATE TABLE IF NOT EXISTS Film (
       numfilm INTEGER NOT NULL PRIMARY KEY,
       titre TEXT,
       annee INTEGER,
       mois INTEGER CHECK (1 <= mois AND mois <= 12),
       realisateur TEXT,
       CONSTRAINT annee_mois_realisateur
                        UNIQUE(annee, mois, realisateur)
     )
```

`CREATE`, syntaxe on forme de diagramme



`CREATE`, syntaxe en forme de Backus-Naur

```
def_table:
    CREATE TABLE [ IF NOT EXISTS ] <nom_table>
    ( <element_de_table> [ { , <element_de_table> }... ] )
    | AS <sous_requete_avec_select>

element_de_table:
    <def_colonne> | <def_contrainte>

def_colonne:
    <nom_colonne> <type_données> <contrainte_de_colonne>

def_contrainte:
   CONSTRAINT <nom_contrainte_table>
    <type_contrainte> <contrainte>
```

- `[ ]` élément optionnel
- `|` alternative
- `{ }` parenthèses
- `...` l'élément précèdent peut être répété

Définition des colonnes

```
def_colonne:
    <nom_colonne> <type_données> <contrainte_de_colonne>
```

- Le nom d'une colonne suit les règles des identificateurs
- `type_données` : un type du SGBD.
  Dans **SQLite3** (parmi d'autres) : `NULL`, `INTEGER`, `REAL`, `TEXT`, `BLOB`.
  Dans **MySQL** : aussi `DATE`, `DATETIME`, `TIME`, `TIMESTAMP`,...
- Contraintes de colonne possibles :

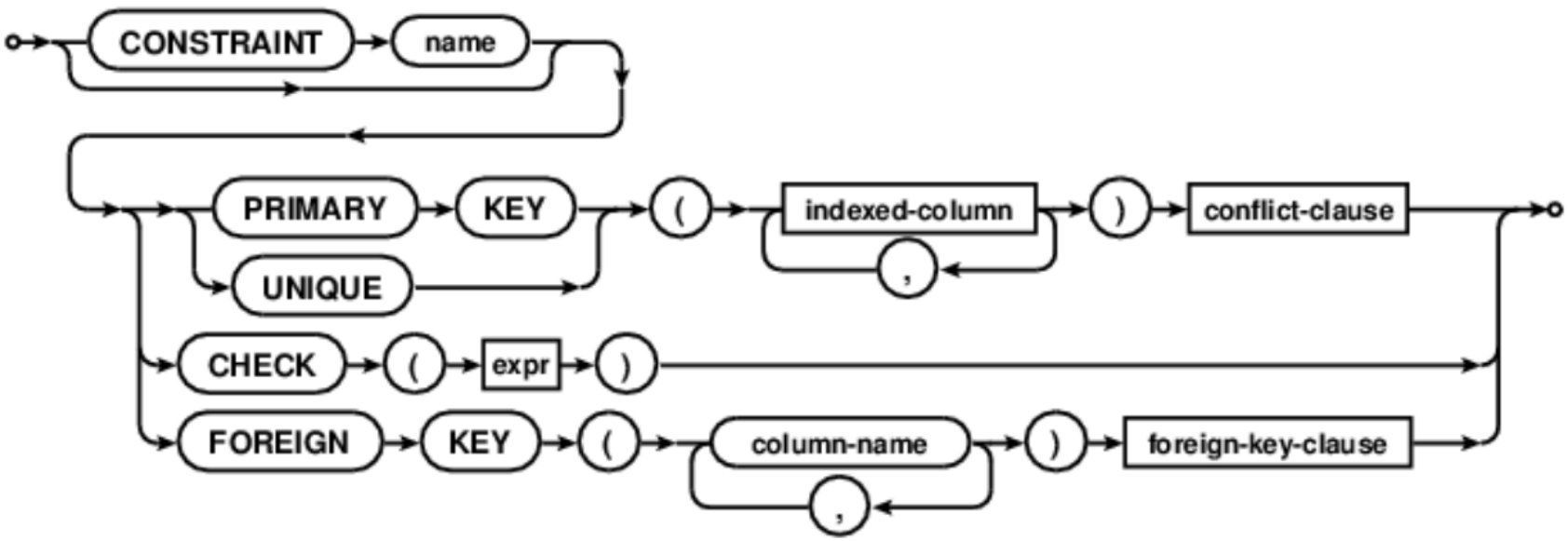
```
NOT NULL

DEFAULT (<exp>)

CONSTRAINT <nom_contrainte>
   { UNIQUE | PRIMARY KEY | CHECK (<exp_booléenne>)
                    | COLLATE NOCASE }
   | REFERENCES <nom_table> [(<noms_colonnes>)]
                          [ON DELETE {CASCADE | SET NULL}]
```

Types de contraintes de colonne

- `UNIQUE` : 2 tuples de cette table ne peuvent avoir la même valeur pour cette colonne.
  Inutile si la colonne a la contrainte de clé primaire.
- `PRIMARY KEY` : spécifie que cette colonne est la clé primaire. Si plusieurs colonnes forment la clé primaire : contrainte de table.
- `CHECK` (exp) : spécifie que les valeurs de cette colonnes doivent vérifier la condition entre parenthèses.
- `COLLATE NOCASE` : pour les chaines de caractères, ne pas tenir compte de la casse.

Contraintes de table

- Apparaissent après la définition des colonnes.
- Nommées.
- Types : même que pour les colonnes.
- Contrainte :
  comme pour les spécifications d'une colonnes,
  met en jeu plusieurs colonnes OU clé étrangère



Contraintes de table : clé étrangère

```
CONSTRAINT <nom_contrainte>
FOREIGN KEY
    ( <nom_colonne_référente> [ ,... ] )
REFERENCES <nom_table_référencée>
    ( <nom_colonne_référencée> [ ,... ] )
[ ON DELETE
    { CASCADE | SET NULL | SET DEFAULT | NO ACTION |RESTRICT} ]
[ ON UPDATE
    { CASCADE | SET NULL | NO ACTION |RESTRICT} ]
```

Définition de contraintes de clé étrangère (suite)

- On peut spécifier ce que l'on fait quand les lignes de la table référencée sont supprimées (`ON DELETE`)
- Supposons que la table A a une clé étrangère portant sur la table B :
  - `CASCADE` : mise à jour et suppression de `B` répercutés sur `A`
  - `SET NULL` : donne la valeur `NULL` aux élément des colonnes de `A` correspondant à celles de t' ayant été touchées.
  - `SET DEFAULT` : donne la valeur de défaut
  - `NO ACTION` : la table n'est pas modifié (action par défaut).
  - `RESTRICT` : empêche la modification qui a amené le changement de la clé étrangère, et génère une erreur.

Création d'une table : exemple

```
table_une(attribut1, attribut2, attribut3)
```

```
CREATE TABLE table_une (
attribut1 INTEGER,
attribut2 INTEGER,
attribut3 TEXT
CONSTRAINT pk_table_une PRIMARY KEY (attribut1, attribut2)
);
```

Création d'une table : exemple

```
CREATE TABLE table_deux (
entier_1 INTEGER PRIMARY KEY,
reel_1 REAL NOT NULL,
entier_2 INTEGER UNIQUE DEFAULT 5,
journee DATE DEFAULT CURRENT_DATE,
entier_3 INTEGER CONSTRAINT c_inf_7 CHECK (entier_3 < 7)
CONSTRAINT c_inf_table_2 CHECK (entier_1 > entier_2),
CONSTRAINT fk_table_2_1 FOREIGN KEY (entier_1, entier_2)
REFERENCES table_une(attribut1, attribut2)
ON DELETE SET NULL
);
```

Requêtes nommées

- À remarquer, `CREATE TMP` avec `AS` :

```
CREATE TMP TABLE maTableTemporaire AS
SELECT ... FROM ... WHERE ...

...

SELECT ... FROM maTableTemporaire WHERE ...
```

- Plus pertinent au LMD.

Modification d'une table

```
ALTER TABLE <nom_table>
[ RENAME TO <nouveau_nom_table> ]
| [ ADD COLUMN <nom_col> <type_données> <contrainte> ]
| [ MODIFY <nom_col> <type_donnée> [DEFAULT val] <contrainte>
| [ DROP COLUMN <nom_col> {CASCADE CONSTRAINTS | INVALIDATE}]
| [ ADD <contrainte_table> ]
| [ DISABLE [UNIQUE] CONSTRAINT <nom_contrainte>
| RENAME COLUMN <old_name> TO <new_name> ]
```

Suppression d'une table

```
DROP TABLE <nom_table>
```

- Que faire des tuples qui faisaient référence à ceux supprimés ?
- Si les contraintes de clés étrangère sont activées, cela lance une requête implicite de délétion et effectue donc les actions prévues (CASCADE, etc.)
- Sinon, le reste de la base n'est pas modifiée.

Consei

- Avec **SQLite3**, utilisez

```
sqlite> .schema
```

pour connaître la définition d'une table.

Plan

*Les instructions **SQL** sur les données*

- Jusqu'à présent :
  manipuler les schémas, c'est-à-dire les tables, les colonnes, les contraintes.
- Dans cette partie :
  - modifier (`INSERT INTO`, `UPDATE ... SET`)
  - supprimer (`DELETE FROM`, `UPDATE ... SET`)
  - interroger (`SELECT ... FROM`, `GROUP BY`, `ORDER BY`)

  les données.

Ajout de données, `INSERT`

```
INSERT INTO <nom_table> [ (<nom_col1>, <nom_col2>, ...) ]
VALUES (<val1>, <val2>, ...);
```

- Respect des types.
- Si les noms de colonnes ne sont pas spécifiées :
  dans l'ordre de définition de la table (déconseillé !).
- Si une colonne n'est pas énoncée,
  sa valeur par défaut lui est attribuée (`NULL` si non spécifiée).
- Si la colonne non-spécifiée est `INTEGER PRIMARY KEY`,
  elle prend la prochaine valeur disponible.
- Les données qui violent des contraintes ne sont pas insérées.
- On peut donner la valeur `NULL` à une colonne.

Ajout de données : exemple

```
INSERT INTO table_deux(entier_1, reel_1, entier_3)
VALUES (12, 6.4, 25);

INSERT INTO table_deux(entier_1, reel_1, entier_2, jour,
 ↪  entier_3)
VALUES (12, 6.4, DEFAULT, DEFAULT, 25);

INSERT INTO table_deux(entier_1, reel_1, entier_3)
VALUES (2, 6, 8);

ERROR !! : ...    Pourquoi ?
```

Car la valeur par défaut de `entier_2` est supérieure à `2` et qu'il existe une contrainte `entier_2 < entier_1`

Modification des données, UPDATE

```
UPDATE <nom_table>
SET {
  <nom_col1> = <val1>, <nom_col2> = <val2>, ...
  |
  (<nom_col1>, <nom_col2>, ...) = (<val1>,<val2>, ...)
}
[ WHERE <condition> ]
```

- les valeurs doivent être des expressions valuables du bon type, ou DEFAULT, ou NULL
- Modifie toutes les lignes de la table respectant la condition du WHERE.

Modification de données : exemples

```
UPDATE hotel SET ville=UPPER(ville);

UPDATE hotel SET etoile=3
WHERE numhotel=4;

UPDATE chambre SET prixnuitht=prixnuitht*1.05
WHERE numhotel IN
(SELECT numhotel
FROM hotel
WHERE ville='AIX');
```

Remarquez la sous-requête dans le `WHERE` :

```
... WHERE numhotel IN (SELECT ... FROM ... WHERE ...);
```

Options de `UPDATE` et `INSERT`

- La syntaxe complète est :

```
UPDATE [OR option] SET colonne=expr [, ...] [WHERE exp]
INSERT [OR option] INTO table(colonne [, ...]) VALUES
 ↪   (expr[, ...])
```

- Options :
  - `IGNORE` : passe à la ligne suivante
  - `FAIL` : arrête le processus
  - `ABORT` : arrête le processus et remet les lignes précédemment modifiées dans leur état initial
  - `REPLACE` : remplace les lignes en contradiction.
    Ex : si `UNIQUE` violée, supprime la ligne correspondante

Suppression de données, DELETE

```
DELETE FROM <nom_table> WHERE <condition>
```

- Pour supprimer toutes les lignes d'une table `T` :

```
DELETE FROM T ;
```

- Répercussions : supprime une ligne référencée par une autre (en clé étrangère).
- Ne pas confondre avec

```
DROP T ;
```

Interrogation de données

Syntaxe de `SELECT` en BNF :

```
SELECT [ DISTINCT ] liste_colonne_resultat
FROM liste_tables [ JOIN name_table ON cond ]
[ WHERE conditions ]
[ START WITH initial_cond ] [ CONNECT BY recusive_cond ]
[ GROUP BY group_by_list [ HAVING search_conditions ] ]
[ ORDER BY order_list [ASC | DESC] ]
```

Interrogation des données

- La commande `SELECT` permet de construire une relation
  - dont les colonnes sont les éléments donnés après la clause `SELECT` (PROJECTION),
  - dont les lignes sont des combinaison de ligne prises dans les tables spécifiées après `FROM` et `JOIN`
  - et qui vérifient les conditions du `WHERE` (RESTRICTION).

Interrogation des données

- `SELECT` : noms de colonnes existantes mais aussi expressions construites sur ces noms (via opérateurs ou fonctions) ou fonction d'agrégat + préfixage si ambiguïtés.
  Renommage via `AS`. Exemple :

```
SELECT avg(nb_emprunt) AS moyenne FROM ...
```

  Attention : `AS` est optionnel, mais utile.

```
SELECT avg(nb_emprunt) moyenne  FROM ...
```

- La clause `DISTINCT` élimine les lignes identiques (doublons) dans le résultat (par défaut `ALL`)
- `FROM` et `JOIN` introduisent les relations impliquées : table nommée mais aussi résultat d'une sous requête nommée.

Interrogation des données

- (`LEFT OUTER`, `INNER`, `NATURAL`) `JOIN`.
  `USING` : pour des jointures sur des colonnes de même noms.

```
SELECT client.numclient, nom, prenom
FROM occupation JOIN chambre ON
( occupation.numchambre = chambre.numchambre
AND occupation.numhotel = chambre.numhotel )
LEFT OUTER JOIN client ON
client.numclient = occupation.numclient
WHERE etage != 1

SELECT client.numclient, nom, prenom
FROM occupation JOIN chambre USING ( numchambre, numhotel )
LEFT OUTER JOIN client USING ( numclient )
WHERE etage <> 1
```

Interrogation des données

- Agrégat
  - `GROUP BY` :
    colonne(s) sur les valeurs desquelles a lieu le regroupement
  - `HAVING` :
    permet d'ajouter un restriction à un calcul d'agrégat
- Exemple : compter le nombre de chambres par ville des hôtels de 3 étoiles ou moins qui possèdent plus de 10 chambres.

```
SELECT ville, COUNT(numchambre)
FROM hotel JOIN chambre USING ( numhotel )
WHERE etoiles <=3
GROUP BY ville
HAVING  COUNT(numchambre) > 10;
```

Interrogation des données

```
ORDER BY col1 [ ASC | DESC ], col2 [ ASC | DESC ]
```

Exemple :

```
SELECT ville, nom
FROM hotel
ORDER BY ville, etoiles DESC;
```

Flow d' un SELECT

1. `FROM ...`
2. `WHERE ...`
3. `GROUP BY ...`
4. `SELECT ...`
5. `HAVING ...`
6. `DISTINCT ...`
7. `ORDER BY ...`
8. `OFFSET ...`
9. `LIMIT ...`

Opérations ensemblistes

```
<requête_1> UNION ALL <requête_2>
<requête_1> UNION <requête_2>
<requête_1> INTERSECT <requête_2>
<requête_1> EXCEPT <requête_2>
```

- Les 2 relations intervenantes doivent avoir des colonnes de même nom et de même type (cf. algèbre relationnelle).
- Note : Pas de gestion des parenthèses à ce niveau !

Plan

Spécificités ***SQLite3***

- Jusqu'à présent :
  norme **SQL** 2003 implémentée en **SQLite3**
- Mais **SQLite3** dévie de la norme sur certains aspects
- Ici : les trucs en plus

Les dates

- Pas de vrai type date : gérer comme des chaines de caractères.
- Création d'une date :
  - `date('YYYY-MM-DD')`
  - `datetime('YYYY-MM-DD HH:MM:SS')`
  - `time('HH:MM:SS')`
  - `Julianday('YYYY-MM-DD')` :
    le nombre de jours depuis le début du calendrier Julien (24 Novembre 4714 B.C.)
- Calcul d'un intervalle de temps en jours :

```
SELECT Julianday(now) - Julianday('1789-07-14')
```

ROWID

- Quand une table est créée, par défaut une colonne (invisible) est ajoutée : `ROWID`
- Agit comme une clé primaire numérique
- Exception : si on définit une colonne de type `INTEGER PRIMARY KEY ASC`
- Pour ne pas en avoir :

```
CREATE TABLE  wordcount(
word TEXT PRIMARY KEY,
cnt INTEGER)
WITHOUT ROWID;
```

Stockage de la base

- La norme est basée sur une architecture client serveur : le serveur stocke la base comme bon lui semble
- La simplification **SQLite3** n'est pas client-serveur
- La base est stockée dans un unique fichier
- La commande shell

  ```
  $ sqlite3 ma_base.db
  ```

  lance **SQLite3** et crée le fichier `ma_base.db` qui contiendra la base.
- Avant de quitter **SQLite3**, la commande `.backup` (de `sqlite3`) sauvegarde la base.

Environnement

- Il existe beaucoup de variable d'environnement :
  - `.help` : aide complète
  - `.import Fichier Table` :
    importe le contenu d'un fichier (par exemple excel) dans une table.
  - `. read Fichier` :
    lit et exécute le code **SQL** contenu dans le fichier.
  - `.mode column|line|csv|html|tcl|...` :
    change le format d'affichage d'une requête `SELECT`

PRAGMA

- Les requêtes `PRAGMA` sont utilisés pour les données internes non-table de la base
- Spécifique à **SQLite3**
- Ne génère pas de message d'erreur
- Syntaxe :

```
PRAGMA nom_pragma
    [= valeur_pragma | (valeur_pragma)] ;
```

- Exemple :

```
PRAGMA foreign_keys ;
PRAGMA foreign_key = ON ;
```

Par défaut c'est `OFF`.

Affichage

- L'affichage par défaut est H.O.R.R.I.B.L.E. !
- On peut y apporter des améliorations :
  - `.header on`
    permet de faire apparaître les noms de colonnes
  - `.mode column`
    aligne correctement les éléments de chaque colonne
  - d'autres fonctionnalités peuvent être utiles : utiliser `.help` pour les connaître.

Plan

Catégories syntaxiques

- La syntaxe **SQL** est divisée en 4 grandes catégories :
  - Identificateurs (noms, variables, . . .)
  - Littéraux (constantes, . . .)
  - Opérateurs
  - Mots réservés
- Ce qui suit : particularités **SQL** pour chaque catégorie.

Analyse lexicale

- Première étape du processus de compilation.
- Objectif :
  transforme un flux (c'est-à-dire, une suite) de caractères en flux d'objets avec structure minimale (appelés jetons).
- On identifie (et on transforme en jeton) les suites de caractères qui sont :
  - identificateurs, c'est-à-dire noms, variables, ...
  - littéraux, constantes, ...
  - opérateurs
  - mots réservés
- Chaque langage de programmation a ses conventions (plutôt règles) lexicales.
- Exemples :
  - Php : les noms des variables debutent par $
  - Haskell : les noms de fonctions debutent par une minuscule,
    ceux des types par une majuscule
  - ...

Identificateurs

- Def. Un identificateur est un nom donné par l'utilisateur à un objet de la base (une table, une colonne, un index, une contrainte, etc.).
- En **SQL** : chaque objet a un identificateur unique à l'intérieur de son espace de noms (et de sa catégorie).
- Les espaces de noms sont hiérarchisés de façon ensemblistes.
- La portée d'un identificateur (c'est-à-dire les espaces où on peut l'utiliser) est son espace de noms, celui qui le contient directement et tous ceux contenus.
- Exemple : on peut avoir des colonnes avec même nom dans des table différentes :

```
client.nom
hotel.nom
```

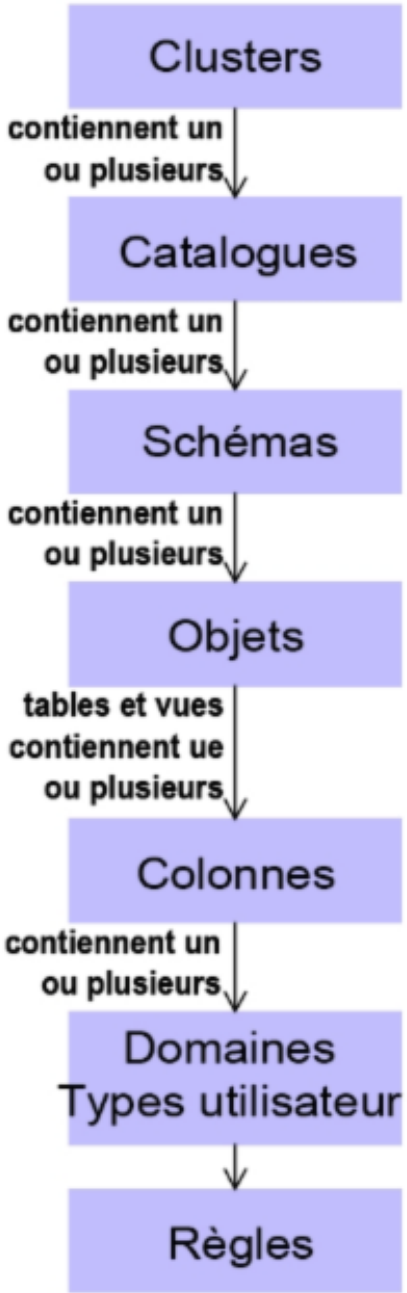
*Règles pour les identificateurs en **SQLite3***

Taille	Pas de limite
Peut contenir	Des chiffres, des caractères ou le blanc souligné
Doit commencer par	Une lettre ou le blanc souligné
Ne peut pas contenir	Des caractères spéciaux (sauf double et simples quotes, à éviter)

Règles spécifiques par SGBDR

Caractéristiques	SGBDR	Spécification
Taille d'un identificateur	SQL2003	128 caractères.
	DB2	128 caractères, dépend de l'objet.
	MySQL	64 caractères.
	Oracle	30 octets (le nombre de caractères dépend du jeu de caractères) ; les noms des bases de données sont limités à 8 octets.
	PostgreSQL	31 caractères (valeur de la propriété NAMEDATALEN moins 1).
	SQL Server	128 caractères (les noms des tables temporaires sont limités à 116 caractères).
Un identificateur peut contenir	SQL2003	Des chiffres, des caractères ou le blanc souligné.
	DB2	Des chiffres, des caractères majuscules ou le blanc souligné.
	MySQL	Des chiffres, des caractères ou des symboles.
	Oracle	Des chiffres, des caractères, le blanc souligné, le signe dièse (#) et le signe dollar ($).
	PostgreSQL	Des chiffres, des caractères ou le blanc souligné.
	SQL Server	Des chiffres, des caractères, le blanc souligné, l'arobase (@), le signe dièse (#) et le signe dollar ($).
Un identificateur doit commencer par	SQL2003	Une lettre.
	DB2	Une lettre.
	MySQL	Une lettre ou un chiffre (mais il ne peut pas être entièrement composé de chiffres).
	Oracle	Une lettre.
	PostgreSQL	Une lettre ou un blanc souligné (_).
	SQL Server	Une lettre, un blanc souligné, une arrobas (@) ou un dièse (#).
Un identificateur ne peut pas contenir	SQL2003	Des espaces ou des caractères spéciaux.
	DB2	Des espaces ou des caractères spéciaux.
	MySQL	Le point (.), le slash (/), les caractères ASCII(0) et ASCII(255). Les apostrophes simples (') et doubles (") ne sont autorisées que dans les identificateurs entre apostrophes.
	Oracle	Des espaces, des apostrophes doubles (") ou des caractères spéciaux.
	PostgreSQL	Des apostrophes doubles (").
	SQL Server	Des espaces ou des caractères spéciaux.

Hiérarchie d'espaces de noms (oracle)



Un cluster est un ensemble de catalogues disponibles pour une session SQL. Il s'agit, en gros, d'une installation du SGBDR. D'après la norme, les clusters contrôlent l'accès aux données et les types de permissions. Notons que Oracle et SQL Server réalisent ces contrôles au niveau des catalogues. Chaque cluster a un nom unique.

Un catalogue est un ensemble de schémas qui persistent au sein du même cluster. En Oracle et SQL Server, on parle d'instance plutôt que de catalogue. Chaque catalogue a un nom unique.

Un schéma est un ensemble d'objets et de données au sein d'un catalogue. Il s'agit, grosso-modo, d'une base de données. Un catalogue doit contenir un schéma de nom INFORMATION_SCHEMA qui contient les meta-données à propos de tous les objets stockés dans le catalogue. Chaque schéma a un nom unique.

Un objet est un ensemble de données ou de fonctionnalités SQL; Parmi les objets courants, citons les tables, les contraintes, les vues, les index, les séquences, les fonctions, etc. Chaque objet a un nom unique. Notons qu'une donnée n'est pas un objet, et ne possède pas de nom.

Une colonne est un ensemble de valeurs identifié par un nom unique (un attribut de relation).

Un domaine (ou un type utilisateur) identifie les valeurs autorisées pour une colonne donnée.

Une règle permet, dynamiquement, de restreindre le domaine de valeurs d'une colonne. Un trigger, par exemple, est une telle règle.

Conventions de nommage

- Pour choisir le nom d'identificateur :
  garantir une lisibilité et simplifier la maintenance.
- Conventions principales :
  - L'identificateur doit être significatif, pertinent et évocateur.
    Exemple : `Acteur` vs `AA_54`
  - Choisir et appliquer la même casse partout
    SQLite3 distingue majuscule et minuscule au niveau des noms.
  - Utiliser les abréviations de manière cohérente
    Exemple : `id` pour toutes les clés primaires numériques
  - Utiliser des noms complets, séparés par '_'
    Exemple : `Date_naissance` vs. `Datenaissance` ou `Datenaiss`

Conventions de nommage (suite)

- Conventions principales (suite) :
  - Ne pas placer de nom “copyrightés” dans un identificateur (nom de société, nom de produit) car ils peuvent évoluer.
  - Ne pas utiliser de préfixes (suffixes) évidents
    Contre-exemple : `BD_CINEMA`.
  - Ne pas utiliser d’identificateur entre double apostrophes (identificateur délimité) car des applications externes peuvent ne pas gérer les caractères spéciaux.

  Les règles sur les identificateurs dépendent des SGBD.

Les littéraux

- Les littéraux sont les valeurs des types élémentaires du système, les constantes.
  (Façon d'écrire les entiers, les réels/flottants, les chaînes de caractères, etc.)
- Littéraux numériques :
  - Les entiers : `321`, `-42`, `0`, `1`, `+20`
  - Les réels :
    - `+88.7` (point vs virgule)
    - `7E3` (=7000)
- Autres littéraux :
  - booléens : `TRUE` et `FALSE`
  - chaînes, entre deux apostrophes verticales simples :
    - `'Hello world'`
    - `'J''aime les BDs'`
  - Les dates : `'2015-01-28 11:21'`

Les opérateurs

- Un opérateur est un symbole (ou un mot court) qui précise une action à effectuer sur des expressions.
- Exemple :
  - `+` est l'opérateur d'addition
  - `AND` est l'opérateur de conjonction logique
- Plusieurs types d'opérateurs :
  - opérateurs de comparaison
  - opérateurs arithmétiques
  - opérateurs logiques
  - opérateurs inclassables

Les opérateurs (suite)

- Les opérateurs sont principalement utilisés par
  le LMD (commandes `SELECT`, `INSERT`, ...)
  et, plus rarement, par
  le LDD (commandes `CREATE TABLE`, `...`)
- Priorités entre opérateurs, à connaître.
- Pas toujours tous implémentés dans le SGBDR :
  il est fortement conseillé de regarder le manuel d'utilisation pour connaître les opérateurs disponibles.
- Pour **SQLite3** :

  `https://www.sqlite.org/lang_expr.html`

Opérateurs arithmétiques

- `+` : addition, concaténation de chaînes
- `-` : soustraction (d'entier, de date, ...)
  Aussi : opérateur unaire de changement de signe
- `*` : multiplication
- `/` : division
- si un opérande est `NULL`, alors le résultat et `NULL`

Opérateurs de comparaison

- Testent si 2 expressions sont :
  - égales, différentes,
  - inférieure l'une à l'autre (pour les types ordonnées)
- Résultat : valeur booléenne `TRUE`, `FALSE`, `UNKNOWN`.
- Si une des expressions a pour valeur `NULL`, le résultat est `NULL`.
- Opérateurs :

=, !=	égalité, inégalité
<, >, <=, >=	comparaisons d'ordre
IN, NOT IN	

Opérateurs logiques

- Utilisés (principalement) dans les clauses `WHERE` et `ON` (de `JOIN`).
- Résultats : valeur booléenne ou `NULL`
- Grand nombre d'opérateurs dans les SGBDR, pas dans la norme.
- Certains parfois classés dans d'autres catégories

Transparent suivant :

g : opérateur d'expression régulière

q : opérateur de sous-requête

t : opérateur sur les lignes ou table

Opérateurs logiques (suite)

Opérateur	*Rq*	*Signification*
AND		TRUE si les deux opérandes valent TRUE
OR		TRUE si l'un des deux opérandes vaut TRUE
NOT		Inverse la valeur de l'opérande ; si l'opérande est un autre comparateur, inverse la valeur finale de la comparaison. Par exemple, NOT LIKE renverra FALSE si le LIKE renvoie TRUE.
ALL	q,t	TRUE si toutes les comparaisons d'un ensemble de comparaisons valent TRUE
ANY	q,t	TRUE si l'une des comparaisons d'un ensemble de comparaisons vaut TRUE
SOME	q,t	TRUE si certaines comparaisons d'un ensemble de comparaisons valent TRUE (équivalent à ANY).
EXISTS	q	TRUE si une sous-requête renvoie des lignes (n'a pas un résultat vide)
LIKE	g	TRUE si l'opérande correspond au motif indiqué. Par exemple, si % est la caractère Joker (qui remplace n'importe quelle chaîne de caractères), l'expression mavar LIKE '%ert%' est évaluée à TRUE si la variable mavar contient le motif (la sous-chaîne) 'ert'. Notons ici que PostGreSQL fournit de surcroît l'opérateur SIMILAR TO qui permet aussi de comparer deux chaînes, dont l'une est une vraie expression régulière.
BETWEEN	q,t	TRUE si l'opérande appartient à l'intervalle spécifié.
IN	q,t	TRUE si l'opérande appartient à l'ensemble d'expressions spécifié, ou si l'opérande est une ligne du résultat de la requête spécifiée.

Autres symboles

Symbole	*Signification*
(et)	Délimiteurs d'expression, pour forcer des priorités.
%	Caractère "Joker" : remplace un caractère (éventuellement vide) ou une chaîne de caractères quelconque, et ce à l'intérieur d'une chaîne de caractères. S'utilise principalement avec l'opérateur LIKE.
_	Dans certains SGBDR, l'underscore est ce caractère Joker qui sert à remplacer exactement un (ni plus ni moins) caractère dans une chaîne. Par exemple '_dd%' dénote une chaîne commençant par un caractère quelconque, suivi de deux caractères 'd', suivis d'une chaîne quelconque de caractères éventuellement vide.
,	La virgule sert de séparateur entre éléments d'une liste (par exemple, une liste de colonnes après un SELECT).
.	Le point sert soit dans la représentation des littéraux réels, soit comme séparateur pour qualifier un identificateur (on préfixe l'identificateur de son espace de noms englobant, par exemple acteur.nom).
--	Délimiteur de commentaire sur une ligne : ce qui suit ce délimiteur, jusqu'à la fin de la ligne, est un commentaire (comme en C)
/* et */	Délimiteur d'un commentaire porté sur plusieurs lignes : ce qui est entre ces délimiteurs est un commentaire (comme en C).
-	Outre la soustraction, ce symbole permet d'exprimer un intervalle de caractères dans une chaîne, notamment dans l'expression de directives CHECK. Par exemple, `ISBN LIKE '[2-4]%'` pour spécifier qu'un numéro ISBN doit commencer par un chiffre entre 2 et 4 (ce qui peut avoir un sens dans une certaine réalité).
;	Fin d'une instruction à exécuter.

Priorité entre opérateurs

- Par ordre de priorité :
  - (et)
  - + et - unaires
  - * et /
  - + et - binaires
  - =, !=, <, >, <=, >=
  - NOT
  - AND
  - ALL, ANY, SOME, BETWEEN, LIKE, OR, IN

Mots clés et réservés

- Exemple : `SELECT`, `CREATE`, `NULL`, `ON`, etc.
- Mot réservé : utilisé ailleurs que le **SQL** par le SGBD (indexation, méta-donnée, constante, etc.)
- Pas comme identificateur : erreur de syntaxe pas facile à déboguer.
- Il faut connaître les mots clés et réservés.

ABORT
ACTION
ADD
AFTER
ALL
ALTER
ANALYZE
AND
AS
ASC
ATTACH
AUTOINCREMENT
BEFORE
BEGIN
BETWEEN
BY
CASCADE
CASE
CAST
CHECK
COLLATE
COLUMN
COMMIT
CONFLICT
CONSTRAINT
COVERING
CREATE
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
DATABASE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DETACH
DISTINCT
DROP
EACH
ELSE
END
ESCAPE
EXCEPT
EXCLUSIVE
EXISTS
EXPLAIN
FAIL
FOR
FOREIGN
FROM
FULL
GLOB
GROUP
HAVING
IF
IGNORE
IMMEDIATE
IN
INDEX
INDEXED
INITIALLY
INNER
INSERT
INSTEAD
INTERSECT
INTO
IS
ISNULL
JOIN
KEY
LEFT
LIKE
LIMIT
MATCH
NATURAL
NO
NOT
NOTNULL
NULL
OF
OFFSET
ON
OR
ORDER
OUTER
PLAN
PRAGMA
PRIMARY
QUERY
RAISE
REFERENCES
REGEXP
REINDEX
RELEASE
RENAME
REPLACE
RESTRICT
RIGHT
ROLLBACK
ROW
SAVEPOINT
SELECT
SET
TABLE
TEMP
TEMPORARY
THEN
TO
TRANSACTION
TRIGGER
UNION
UNIQUE
UPDATE
USING
VACUUM
VALUES
VIEW
VIRTUAL
WHEN
WHERE
WITH
WITHOUT